UZL-Testbed User Guide

FRONTS
Foundations of Adaptive Networked Societies of Tiny Artefacts

www.fronts.cgi.gr

Document history

Version	*Date*	*Changes*
1.0	01.08.2009	Initial version
1.1	05.10.2009	iShell documentation added
1.2	31.05.2010	Revised

Contents

1. About this User Guide 4
2. General Description of the UZL-Testbed 5
3. Simulating the UZL-Testbed 6
4. Using the testbed 11
    4.1. Connecting a local iShell2 to the gateway node 0x94a0 12
    4.2. Flashing the gateway node 13
    4.3. Flashing the network over the air 15
    4.4. Closing the connection 17

1. About this User Guide

In this user guide,

- files and folders are represented in the Arial typeface,
- code fragments, function names etc. are represented in the `Courier New` typeface,
- GUI elements such as button descriptions etc. are represented in “quotation marks”,
- titles of other documents are presented in *Italic* type.

This manual assumes that the reader has successfully installed the iSense development environment, and obtained the iSense standard firmware. For further information on theses steps, consult the *Development Environment Setup User Guide* [1].

In addition, it is assumed that the user is familiar with the use of iShell. For further information on iShell, consult the *iShell User Guide* [2].

For further information on iSense firmware programming concepts and on application development, it is recommended to read the *Writing iSense Applications User Guide* [3].

2. General Description of the UZL-Testbed

This user guide describes

- how the UZL-testbed looks like, and
- how an application can be simulated in Shawn using the same environment parameters.

So far, the testbed comprises the 22 iSense sensor nodes (of type R1) shown in Table 1: MAC addresses of UZL-testbed nodes.. Two of them are so-called gateway nodes equipped with a Core Module (cf. [4]) with SMA antenna and a Gateway Module (cf. [5]) connected to a computer. The Gateway Modules enable power supply and communication of the node to iShell running on the connected computer and vice versa. The Gateway Module can also be used to program the connected Core Module.

The ordinary sensor nodes are equipped with a Core Module with SMA antenna and a Security Module including a PIR (passive infrared) sensor and an accelerometer. They are plugged to power outlets. Technical data, software documentation and a demo application concerning the Security Module can be found in [6].

MAC (Hex)	MAC (Dec)	Type
06 0D	1549	
06 0e	1550	
06 14	1556	
06 17	1559	
06 35	1589	
06 40	1600	
08 78	2168	
08 7f	2175	
78 54	30804	
78 56	30806	
78 57	30807	
78 5c	30812	
78 68	30824	
78 6a	30826	
94 73	38003	
94 7d	38013	
94 7e	38014	
94 7f	38015	
94 83	38019	
94 9e	38046	
94 a0	38048	Gateway
94 a4	38052	Gateway

Table 1: MAC addresses of UZL-testbed nodes.

A node is identified uniquely by its MAC address. The first column of Table 1 shows the MAC addresses as hexadecimal, the second column as decimal value. For example, using the command

```
os().debug("My id is %x = %d.", os().id(), os().id())
```

results in the following output in the SerialMonitor-Plugin of iShell:

```
My id is 0x94a0 = 38048.
```

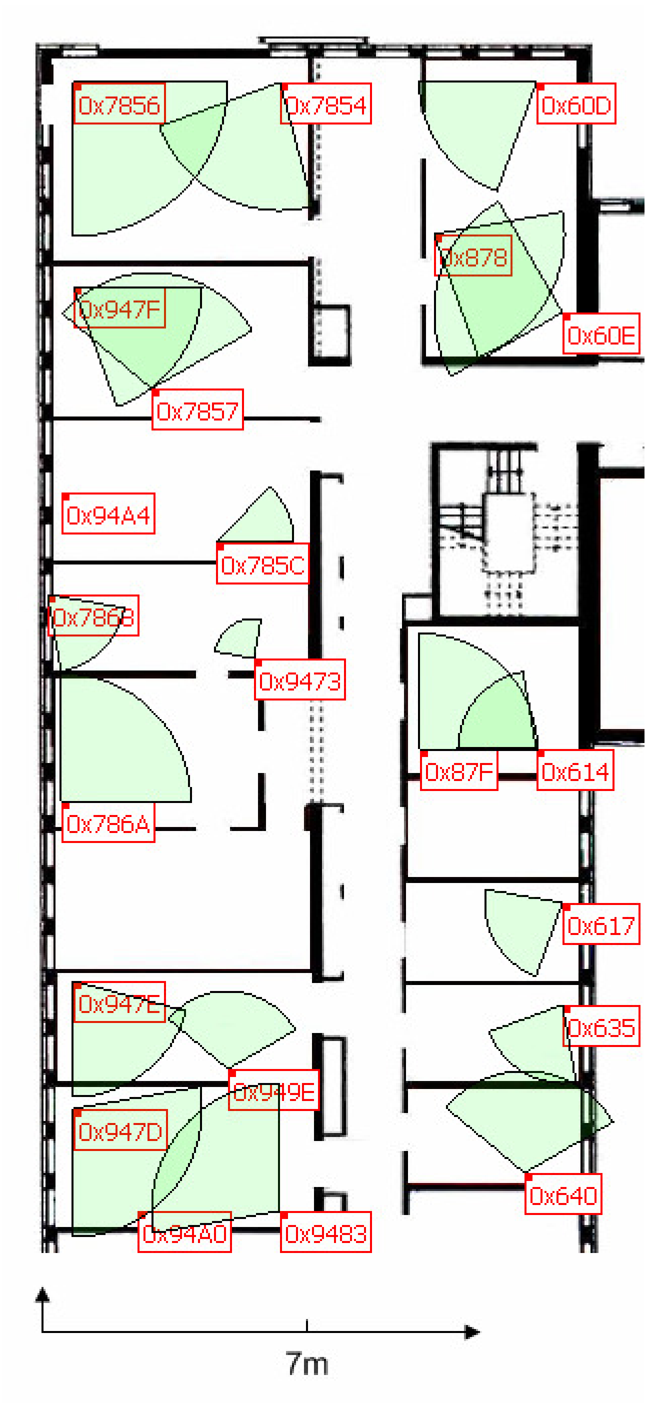


Figure 1: Deployment of the sensor nodes within the Institute of Telematics

Figure 1 shows the spatial arrangement of the sensor nodes within the Institute of Telematics in Lübeck. The sensing ranges of the PIR sensors are indicated by light green cones. The range of such a sensor goes up to 10m in a free area, but walls and furniture limit them literally in indoor scenarios. The two gateway nodes are not equipped with a PIR sensor.

3. Simulating the UZL-Testbed

This section describes how simulations can be run with the parameters of the UZL-testbed. We assume that the reader is used to Shawn configuration files to control a Shawn simulation run. As JShawn offers scripting functionality and the use of local variables we use JShawn to configure the simulation. Refer to [7] for detailed information.

The testbed topology is described by /scenario_fronts.xml. The xml-file defines the real positions of the nodes as well as the link properties between nodes.

As the simulator uses increasing numbers from 0 to n as ids for n+1 nodes for an efficient algorithmic processing we added functionality in **iSense** enabling the usage of arbitrary ids. Setting the global variable `use_tag_ids` to "true" in your configuration file like

```
shawn.setGlobalVariable("use_tag_ids", "true");
```

causes that iSense ids are read from a node tag called "isense_id" which is also defined in the xml-file.

Note, that os().id() has not necessarily the same value as os().proc().owner().id() in this case!!

To find a Shawn::Node by its iSense MAC address (os().id()) use the method

```
class ShawnGlueProcessor
{
  static shawn::Node* find_node_by_id_w(shawn::World& w_,uint16 i)throw();
}
```

instead of owner().world().find_node_by_id(1234).

```
Example: ShawnGlueProcessor::find_node_by_id_w((ShawnOs* &os_)->proc().
      owner_w().world_w(), 0x060d);
```

The above mentioned xml-file contains these ids additionally to the positions:

```
<scenario>
      <snapshot id="0">
            …
            <node id="19"><location x="3.0" y="2.6" z="0.0"/>
                  <tag type="int" name="isense_id" value="38048"/>
                                                      <!-- Hex: 94 a0 -->
            </node>
            …
      </snapshot>
</scenario>
```

You can load this snapshot by using the shawn command `load_world`:

```
shawn.runCommand("load_world", "world_in_file=scenario_fronts.xml
                                       snapshot=0 processors=isense");
```

Table 2: Real and Shawn MAC addresses shows the Shawn ID for each node which is returned by os().proc().owner().id(). Normally, the application should neither know nor use them. iSense hides them completely.

MAC (Hex)	Shawn ID
06 0D	0
06 0e	1
06 14	2
06 17	3
06 35	4
06 40	5
08 78	6
08 7f	7
78 54	8
78 56	9
78 57	10
78 5c	11
78 68	12
78 6a	13
94 73	14
94 7d	15
94 7e	16
94 7f	17
94 83	18
94 9e	19
94 a0	20
94 a4	21

Table 2: Real and Shawn MAC addresses

The JShawn-file /configuration.jshawn contains all commands to execute one simulation run with the UZL-testbed parameters. A simulation includes generating the world, establishing a connection to iShell if required, enabling logging tasks and running the simulation.

```
//------------------------------------------------
// Simulation parameters
//------------------------------------------------
// Determines if the simulation shall wait to be connected to iShell
boolean connect_to_ishell = true;

// Number of executed iterations
int iterations = 5000;

// The random seed. Use something greater or equal zero to enable the seed
int seed = 1;
```

The link qualities between nodes are defined in the environment. To enable these you must

- set the variable "env_config" in your xml-file which contains the environment when you create the Shawn world (with the prepare_world-command) and
- use the Shawn "linkprobability" communication model. Here, each (unidirectional) communication link is defined by setting its probability for a successful packet transmission. A link probability of 0.5 means that every second packet is lost.

```
//-------------------------------------------------
// Shawn models and scenario parameters
//-------------------------------------------------

// Shawn edge model
String em = "simple";

// Shawn communication model
String cm = "link_probability";

// Shawn transmission model
String tm = "csma";

// File containing the UZL testbed positions and ids
String scenario = "scenario_fronts.xml";

//Set the seed
if( seed > 0 )
    shawn.runCommand("random_seed", "action=set seed=" + seed);

// Use the ids of the fronts testbed instead of 0 to 21
shawn.setGlobalVariable("use_tag_ids", "true");

int range = 35;

//Create a new simulation world and set its parameters
shawn.runCommand("prepare_world","edge_model=" + em +
    " comm_model=" + cm + " range=" + range + " size_hint=" + range +
    " transm_model=" + tm + " bandwidth=16000 csma_type=802.15.4
      immediate_delivery=true" + " env_config=" + scenario);

shawn.setGlobalVariable("count","22");

// Load the fronts scenario
shawn.runCommand("load_world", "world_in_file="+scenario+" snapshot=0
processors=isense");
```

You can run simulations with and without connecting Shawn to iShell. To connect Shawn to iShell use the ShawnSocketTask as follows.

```
//-------------------------------------------------
// Connection to iShell
//-------------------------------------------------

if (connect_to_ishell)
    shawn.runCommand("ShawnSocketTask","socket_port=1280
                                               socket_blocking=true");
```

For each ShawnSocketTask in the JShawn-file open an iShell instance. In iShell, the *address* has to be set to *localhost* and the *port* to the value set in the JShawn-file (1280 in the example). The *Main node* connects iShell to one node. If your iSense application implements the UartPacketHandler interface and is registered for at least one packet type, the iShell-Messenger-Plugin can send (uart) messages to this node. **Since node 0x94a0 is the gateway node in the UZL-testbed, set "38048" as main node.** The range *debug nodes* in iShell specifies the set of nodes from which messages will be transmitted to the iShell. "*" means all nodes. "2157, 1549" means nodes 0x87f and 0x60d. They can be set to arbitrary values. If you run the application their debug output will be shown in the *Serial Monitor* plugin. Note, that it depends on the value of `use_tag_ids` if the numbers from 0 to n or the decimal values of the MAC addresses must be used for a successful communication between Shawn and iShell.

Shawn offers the possibility of tracing radio messages and fill levels of queues in files while the simulation is running. The following values for parameter *create* are implemented:

- postscript_messages
- msg_density_postscript
- postscript_queues
- postscript_timeout_task_queues
- postscript_memory
- gnuplot_memory

```
//-------------------------------------------------
// Logging parameters (tracing in files)
//-------------------------------------------------

// Trace every single message in the network into a .ps file
//shawn.runCommand("isense_trace", "create=postscript_messages
file=messages.ps message_delta=1");


// Trace the message density on the links of the network into a .ps file
//shawn.runCommand("isense_trace", "create=msg_density_postscript
file=msg_density.ps time_delta=100 max_width=3");

// Trace incomming and outgoing radio queues of the nodes into a .ps file
//shawn.runCommand("isense_trace", "create=postscript_queues
file=radio_queues.ps time_delta=1.0");

// Trace the timeout and task queues of the nodes into a .ps file
//shawn.runCommand("isense_trace", "create=postscript_timeout_task_queues
file=timeout_queues.ps time_delta=1.0");

// Trace the heap usage of the nodes into a .ps file
//shawn.runCommand("isense_trace", "create=postscript_memory
file=memory.ps time_delta=1.0");

// Trace the detailed heap usage of the nodes into a gnuplot data file
//shawn.runCommand("isense_trace", "create=gnuplot_memory fmalloc=1
ffree=1 nodes=* file=mem-of-node-");
```

Finally, run the simulation task to launch the execution of the simulation.

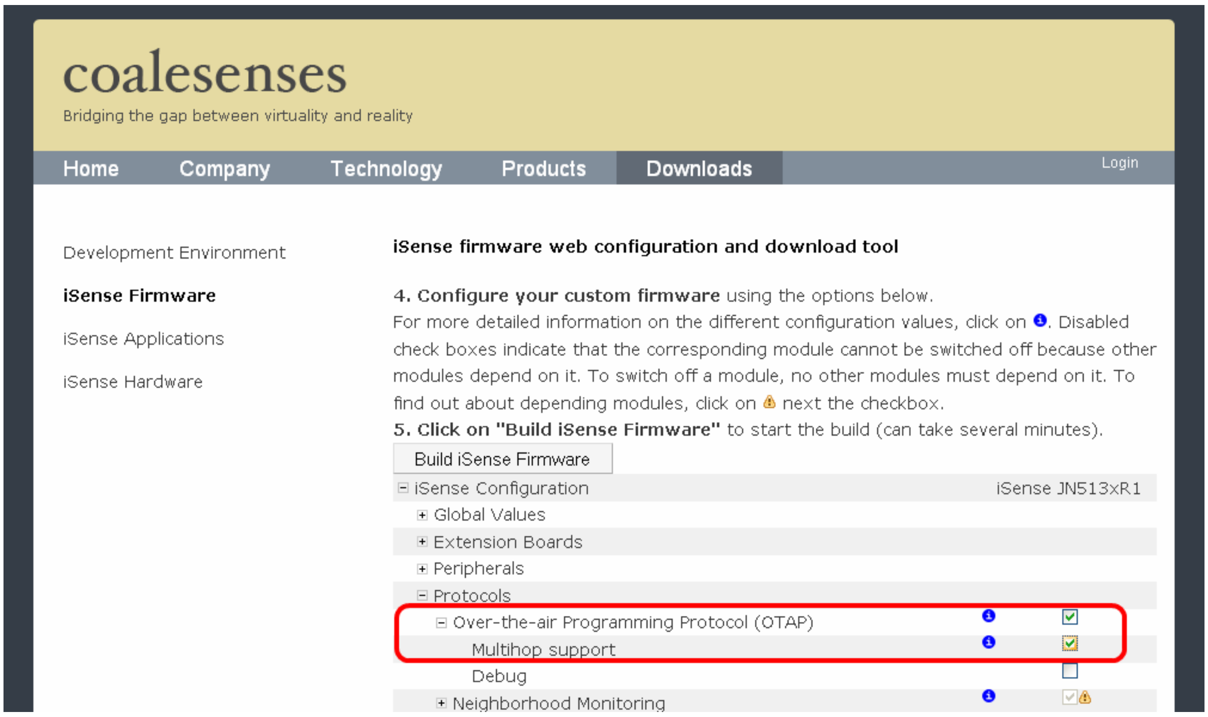
```
//-------------------------------------------------
// Execution
//-------------------------------------------------

// Run the simulations
shawn.runCommand("simulation", "max_iterations="+iterations);
```

4. Using the testbed

Before you flash your application on the nodes ensure

- that your application does not sleep at all or that the device wake up from time to time so that they can be reprogrammed (e.g., by calling the command `os().allow_sleep(false);` in the boot-method),
- that your application runs on radio channel 21 (by calling the command `os().hardware_radio().set_channel(21);` in the boot-method),
- that your application can be stopped somehow if it sends many messages, because otherwise a reprogramming becomes impossible. The TopologyRecognition application [8] uses for example the flooding protocol to send a STOP-command to all nodes which sets a boolean variable SEND to false, and
- that the iSense firmware version that you use contains **over the air programming with multihop support**, so that the testbed remains programmable while your application is running. Therefore, you have to enable the check box "Multihop support" in ->Protocols->Over-the-air Programming Protocol (OTAP) when building the iSense firmware (indicated by the red box in the following screenshot).

Important: If you are unsure if devices running your application are programmable over the air do not immediately program the entire network! Program only one node and reprogram it again to test if it works fine. Then, program all other nodes.



iSense firmware web configuration and download tool

4. Configure your custom firmware using the options below.
For more detailed information on the different configuration values, click on ●. Disabled check boxes indicate that the corresponding module cannot be switched off because other modules depend on it. To switch off a module, no other modules must depend on it. To find out about depending modules, click on ⚠ next the checkbox.
5. Click on "Build iSense Firmware" to start the build (can take several minutes).

```
⊟ iSense Configuration                                  iSense JN513xR1
  ⊞ Global Values
  ⊞ Extension Boards
  ⊞ Peripherals
  ⊟ Protocols
    ⊟ Over-the-air Programming Protocol (OTAP)     ●      ✓
        Multihop support                           ●      ✓
        Debug                                             ☐
    ⊞ Neighborhood Monitoring                      ●      ✓⚠
```

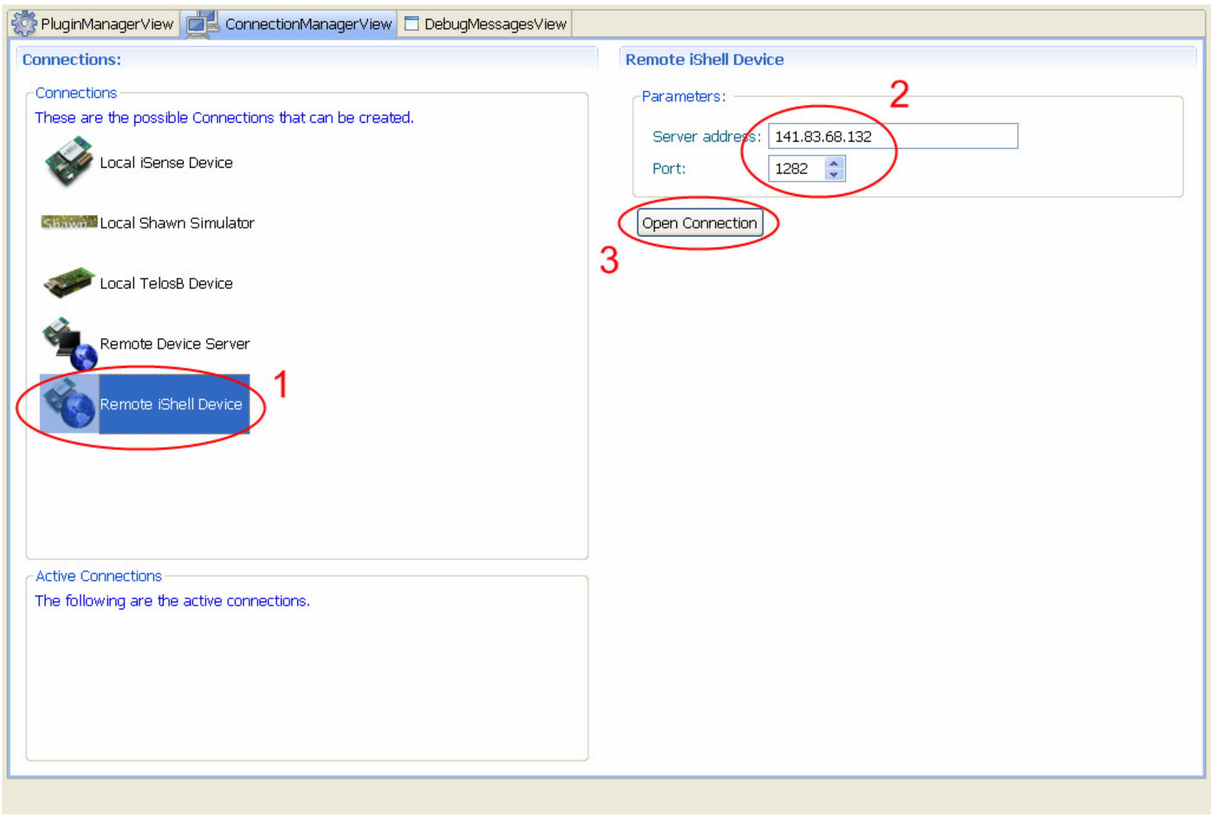
4.1. Connecting a local iShell2 to the gateway node 0x94a0

First tell somebody in Lübeck who is in touch with the testbed (e.g. via email), that you want to use the testbed and for how long you will probably need it. If nobody else is using the testbed, this person will start the server to which you can connect your client. If a test is currently running on the testbed, the person will tell you when it will be free again.

To start an iShell on your computer:

- Download ishell2.zip from [8] and unzip it.
- Start ishell.exe.

Then, open the *ConnectionManagerView* and click on *Remote iShell Device* (1). Enter the server address 141.83.68.132 as well as the port number 1282 (c.f. 2). Now, click on *Open Connection* (3). The connection will be established. If there is a problem with that, let somebody in Lübeck know about it.



These are the possible Connections that can be created.

The following are the active connections.

Server address: 141.83.68.132

Port: 1282

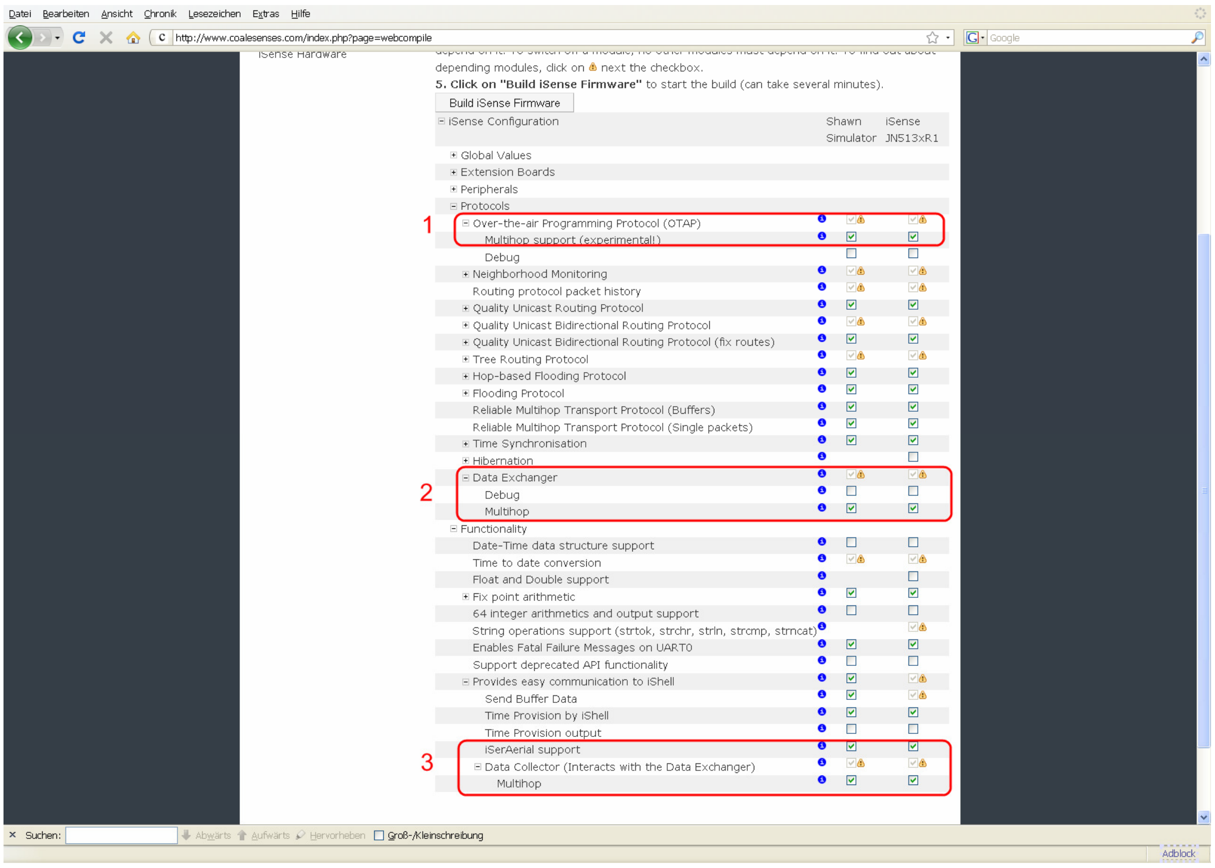
4.2. Flashing the gateway node

Note, that you have to flash the gateway node (using the iShell Flash Loader Plugin) separately from all other nodes which are programmed over the air. Either program the gateway node with a so-called *Data-Collector-and-iSeraerial* application (you can find the corresponding bin-file at [8]) or enable the iSeraerial including multihop support functionality of iSense in your application by

1. selecting iSeraerial support (first line in the red box 3)
2. constructing an `IShellInterpreter` in your application and calling `enable_seraerial()` on it:

```
IShellInterpreter *isi_ = new IShellInterpreter(os());
isi_->enable_seraerial();
```

If you want to the gateway node to collect the results of your test via the data exchange protocol enable the `DataExchanger` (red box 2). It collects the data of all registered `DataProviders` using the tree routing protocol. For the data collection the gateway node must run the above mentioned *Data-Collector-and-iSeraerial application* or you must enable the *Data Collector -> Multihop* support (last line in box 3).

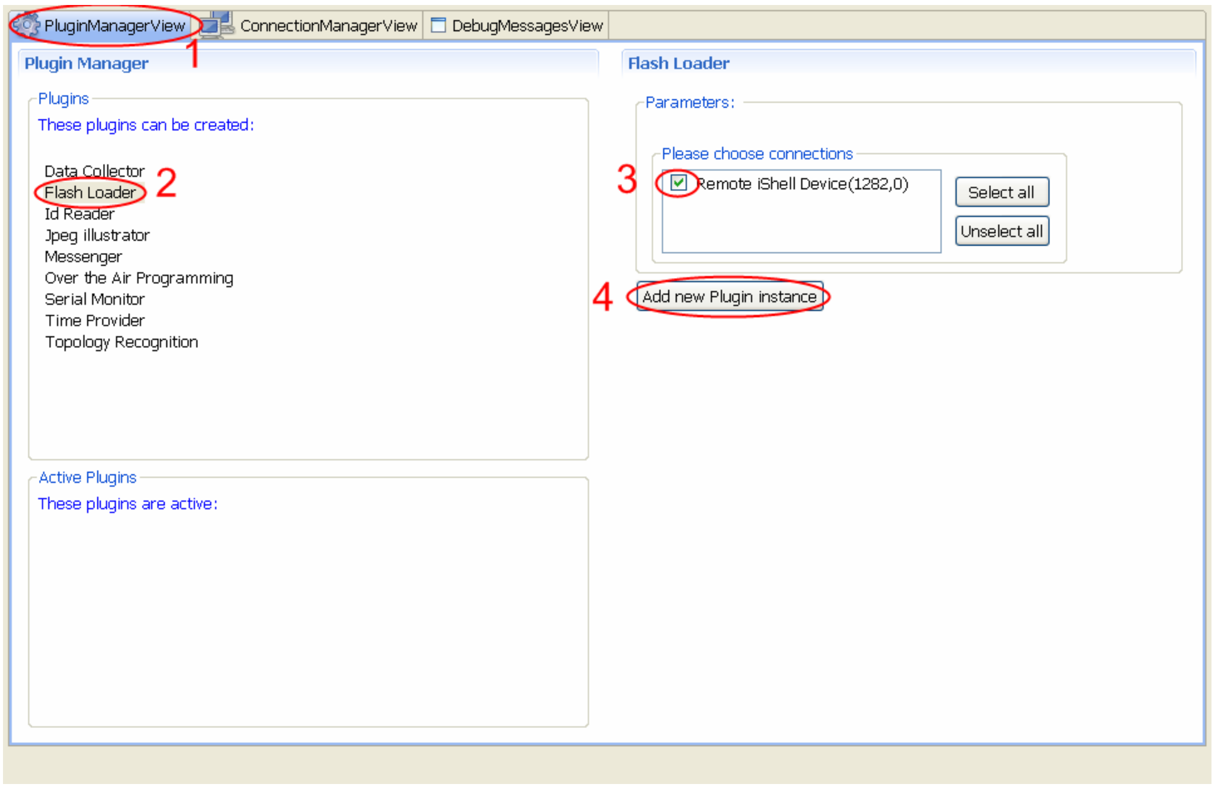

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://www.coalesenses.com/index.php?page=webcompile
Google
iSense Hardware
Suchen: Abwärts Aufwärts Hervorheben Groß-/Kleinschreibung
Adblock

depending modules, click on ⚠ next the checkbox.
5. Click on "Build iSense Firmware" to start the build (can take several minutes).

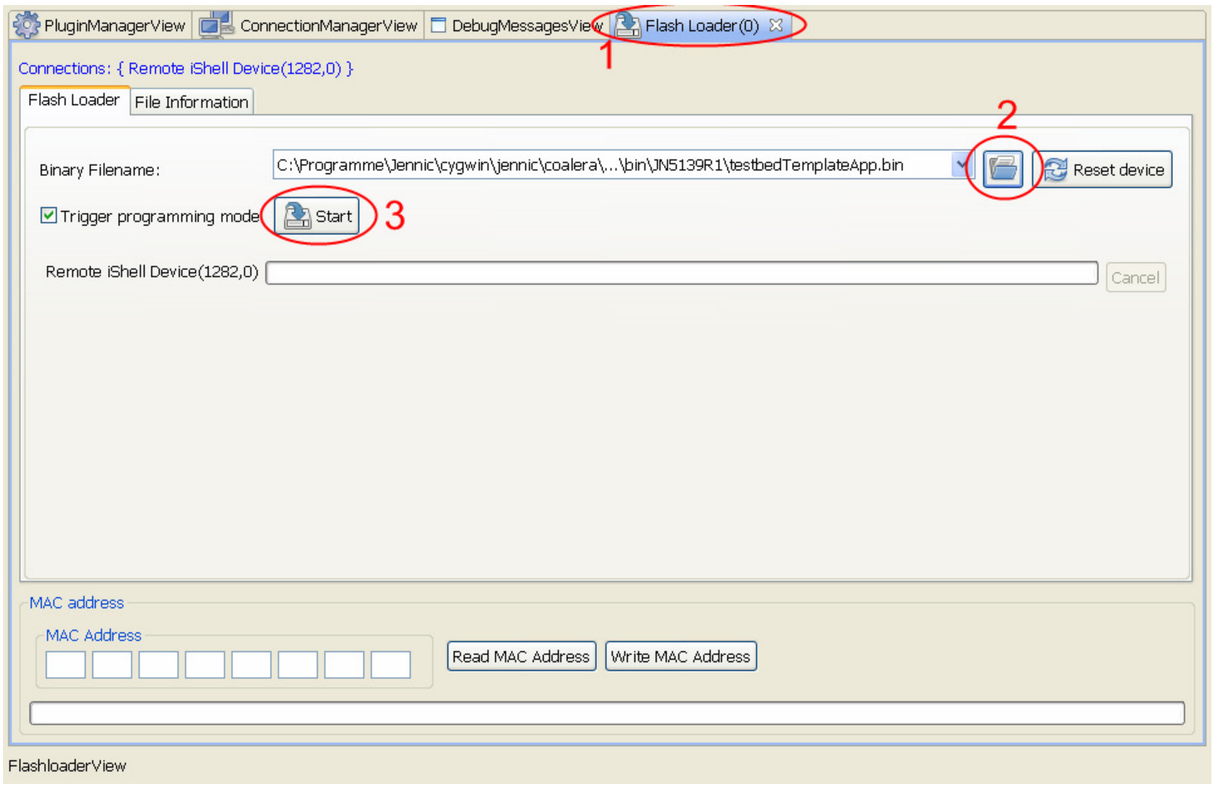
Build iSense Firmware

	iSense Configuration	Shawn Simulator	iSense JN513xR1
	⊞ Global Values		
	⊞ Extension Boards		
	⊞ Peripherals		
	⊟ Protocols		
1	⊟ Over-the-air Programming Protocol (OTAP)	✓⚠	✓⚠
	Multihop support (experimental!)	✓	✓
	Debug	☐	☐
	⊞ Neighborhood Monitoring	✓⚠	✓⚠
	Routing protocol packet history	✓⚠	✓⚠
	⊞ Quality Unicast Routing Protocol	✓	✓
	⊞ Quality Unicast Bidirectional Routing Protocol	✓⚠	✓⚠
	⊞ Quality Unicast Bidirectional Routing Protocol (fix routes)	✓	✓
	⊞ Tree Routing Protocol	✓⚠	✓⚠
	⊞ Hop-based Flooding Protocol	✓	✓
	⊞ Flooding Protocol	✓	✓
	Reliable Multihop Transport Protocol (Buffers)	✓	✓
	Reliable Multihop Transport Protocol (Single packets)	✓	✓
	⊞ Time Synchronisation	✓	✓
	⊞ Hibernation		☐
2	⊟ Data Exchanger	✓⚠	✓⚠
	Debug	☐	☐
	Multihop	✓	✓
	⊟ Functionality		
	Date-Time data structure support	☐	☐
	Time to date conversion	✓⚠	✓⚠
	Float and Double support		☐
	⊞ Fix point arithmetic	✓	✓
	64 integer arithmetics and output support	☐	☐
	String operations support (strtok, strchr, strln, strcmp, strncat)		✓⚠
	Enables Fatal Failure Messages on UART0	✓	✓
	Support deprecated API functionality	☐	☐
	⊟ Provides easy communication to iShell	✓	✓⚠
	Send Buffer Data	✓	✓⚠
	Time Provision by iShell	✓	✓
	Time Provision output	☐	☐
3	iSerAerial support	✓	✓
	⊟ Data Collector (Interacts with the Data Exchanger)	✓⚠	✓⚠
	Multihop	✓	✓

- To flash the gateway node open a flash loader (2) connected to the Remote iShell Device (3).



- Switch to the generated Flash loader Plugin instance (1).
- Then, select the bin-file you want to flash on the node (2).
- Click on the Start button (3) and wait until the flash process has been finished.

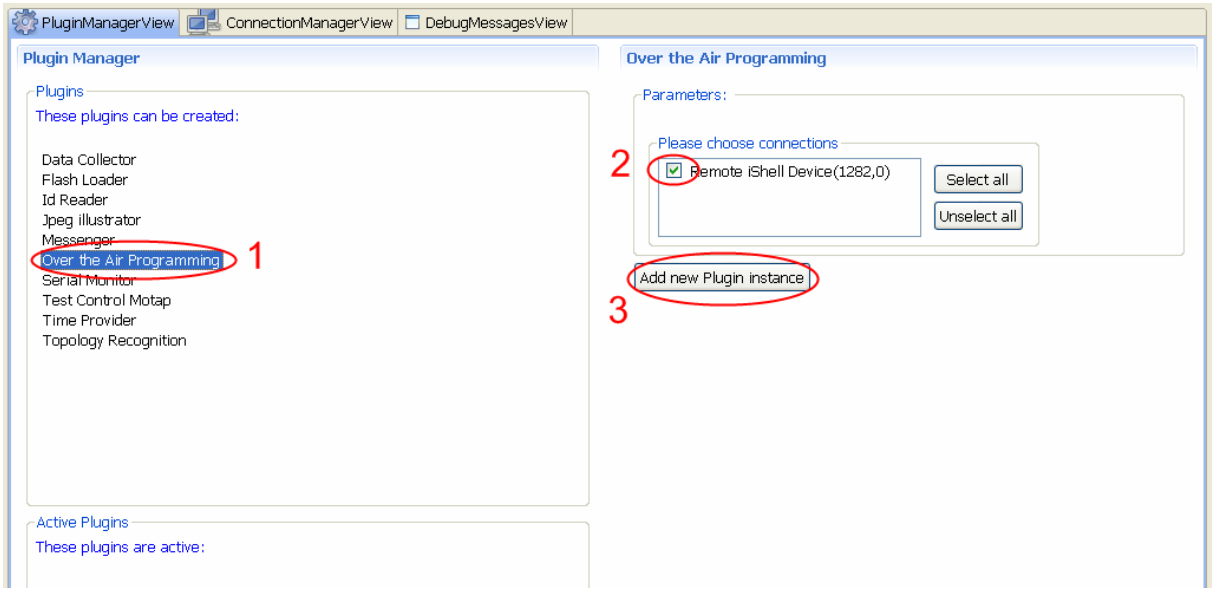

Connections: { Remote iShell Device(1282,0) }

Binary Filename: C:\Programme\Jennic\cygwin\jennic\coalera\...\bin\JN5139R1\testbedTemplateApp.bin

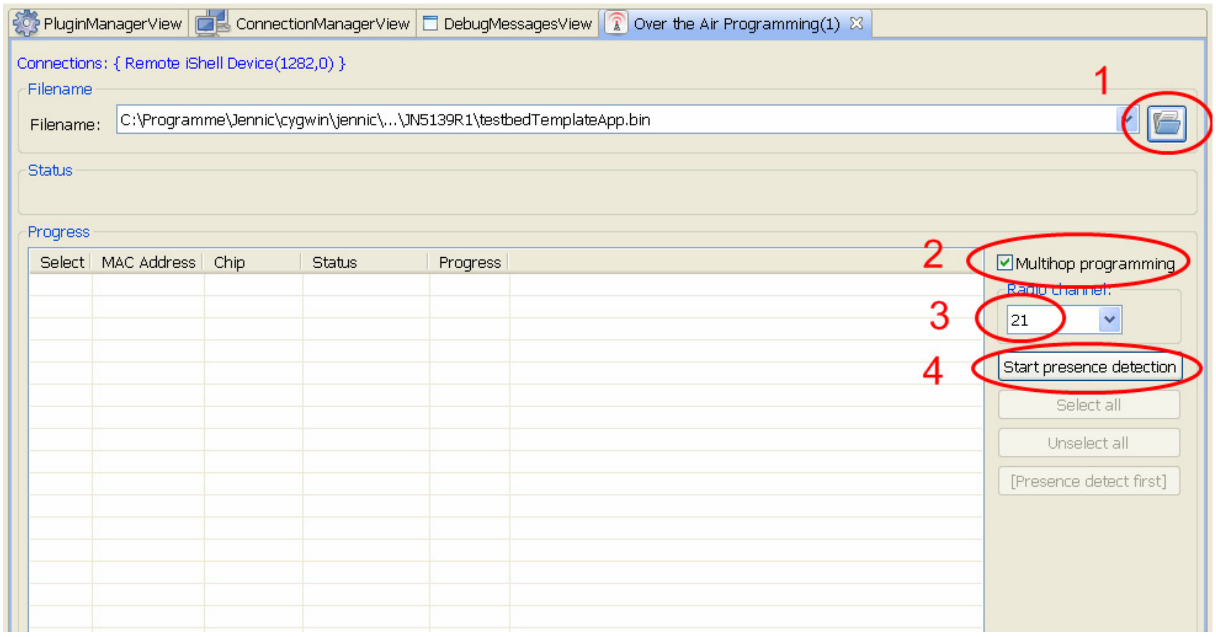
☑ Trigger programming mode

4.3. Flashing the network over the air

- First, open the *PluginManagerView*.
- Then, click on *Over the air Programming* (1).
- Select the *remote iShell Device* (2) and
- click on *Add new Plugin instance* (3) to generate an instance of this plugin.

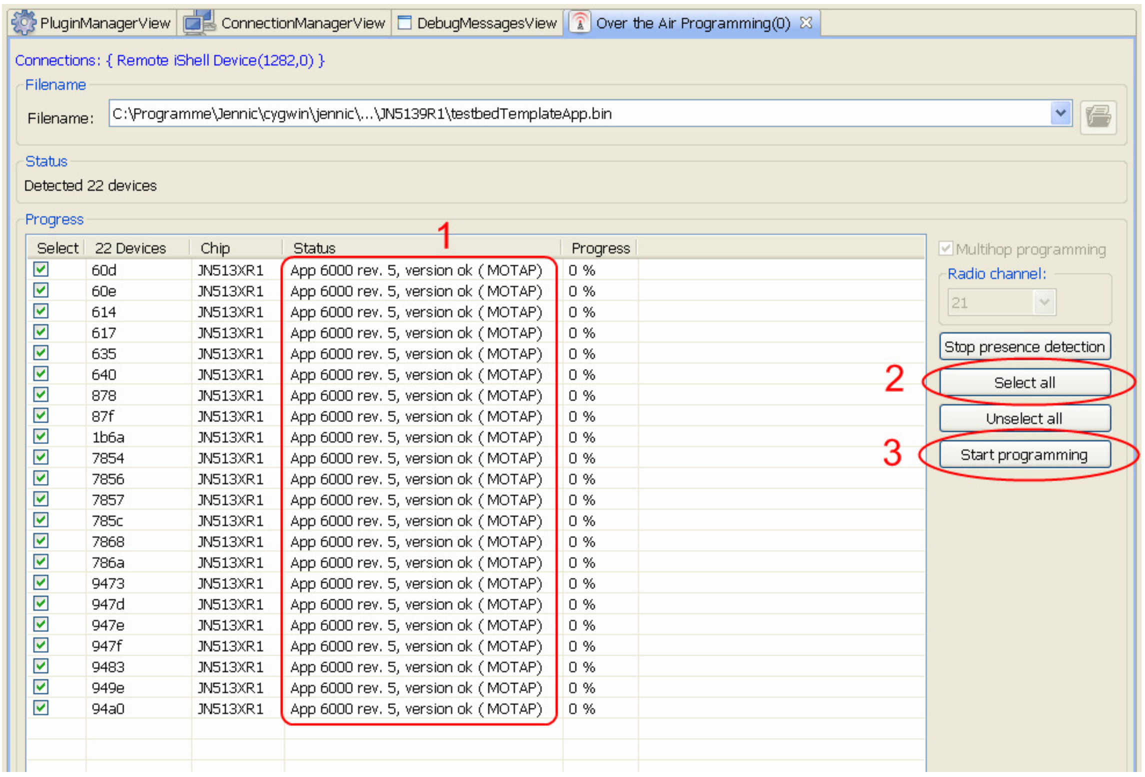


- Select the new plugin instance.
- Select the bin-file containing your program which you want to run on the testbed (1).
- Click on *Multihop programming* (2).
- Select radio channel 21 as this is the channel of FRONTS users in Lübeck (3).
- Click on Start presence detection (4).



The *Status* column (1) shows the application identification and software revision number of the currently running application. They can be used to verify that the correct program is running after the flashing. (To do this verification, run the presence detection after the flashing for a short time and stop it after the devices appeared in the list.)

- When 21 devices are detected, click on Select all (2) and Start programming (3).



Connections: { Remote iShell Device(1282,0) }

Filename

Filename: C:\Programme\Jennic\cygwin\jennic\...\JN5139R1\testbedTemplateApp.bin

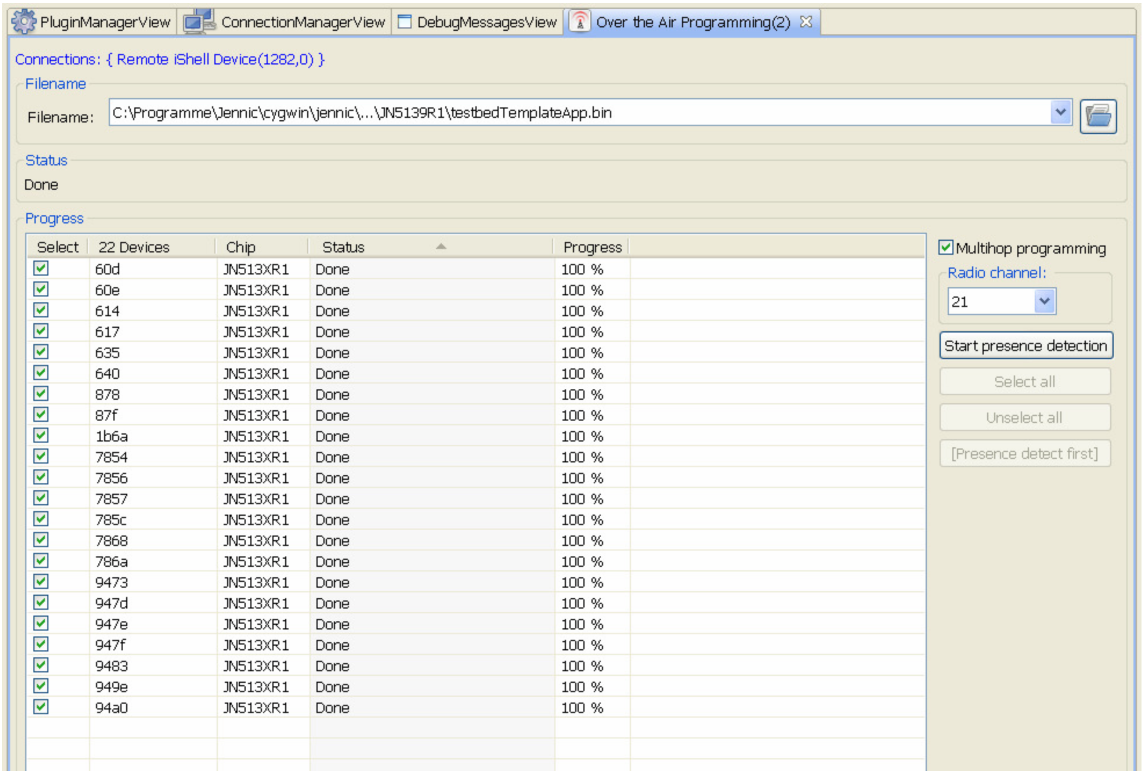
Status

Detected 22 devices

Progress

Select	22 Devices	Chip	Status	Progress
✓	60d	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	60e	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	614	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	617	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	635	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	640	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	878	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	87f	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	1b6a	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	7854	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	7856	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	7857	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	785c	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	7868	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	786a	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	9473	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	947d	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	947e	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	947f	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	9483	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	949e	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %
✓	94a0	JN513XR1	App 6000 rev. 5, version ok (MOTAP)	0 %

- Wait until the *Status* is *Done*. Then, you can start your test.



Connections: { Remote iShell Device(1282,0) }

Filename

Filename: C:\Programme\Jennic\cygwin\jennic\...\JN5139R1\testbedTemplateApp.bin

Status

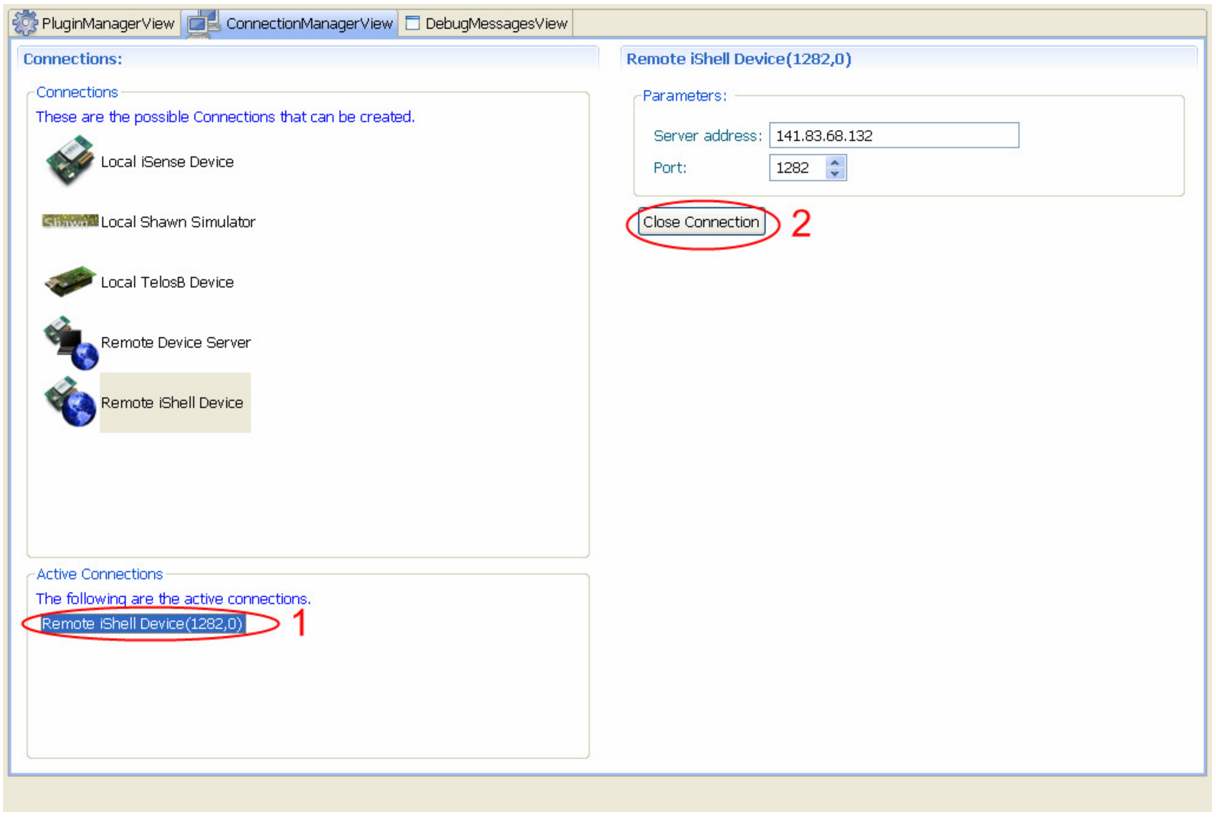
Done

Progress

Select	22 Devices	Chip	Status	Progress
✓	60d	JN513XR1	Done	100 %
✓	60e	JN513XR1	Done	100 %
✓	614	JN513XR1	Done	100 %
✓	617	JN513XR1	Done	100 %
✓	635	JN513XR1	Done	100 %
✓	640	JN513XR1	Done	100 %
✓	878	JN513XR1	Done	100 %
✓	87f	JN513XR1	Done	100 %
✓	1b6a	JN513XR1	Done	100 %
✓	7854	JN513XR1	Done	100 %
✓	7856	JN513XR1	Done	100 %
✓	7857	JN513XR1	Done	100 %
✓	785c	JN513XR1	Done	100 %
✓	7868	JN513XR1	Done	100 %
✓	786a	JN513XR1	Done	100 %
✓	9473	JN513XR1	Done	100 %
✓	947d	JN513XR1	Done	100 %
✓	947e	JN513XR1	Done	100 %
✓	947f	JN513XR1	Done	100 %
✓	9483	JN513XR1	Done	100 %
✓	949e	JN513XR1	Done	100 %
✓	94a0	JN513XR1	Done	100 %

4.4. Closing the connection

Open again the *ConnectionManagerView* and select the Remote iShell connection to the gateway node (1) and close the connection via button 2.



These are the possible Connections that can be created.

The following are the active connections.

Remote iShell Device(1282,0)

Server address: 141.83.68.132

Port: 1282

References

[1] coalesenses Development Environment Setup User Guide, online available at http://www.coalesenses.com/download/UG_development_environment_setup_v1.9_web.pdf

[2] coalesenses iShell User Guide, online available at http://www.coalesenses.com/download/UG_ishell_v1.3.pdf

[3] coalesenses Writing iSense Applications User Guide, online available at http://www.coalesenses.com/download/UG_writing_isense_applications_1v1.pdf

[4] coalesenses iSense Core Module User Guide, online available at http://www.coalesenses.com/download/UG_CM10X_1v0.pdf

[5] coalesenses iSense Gateway Module User Guide, online available at http://www.coalesenses.com/download/UG_GM10X_1v0.pdf

[6] coalesenses iSense Security Module User Guide, online available at http://www.coalesenses.com/download/UG_SM10AX_1v0.pdf

[7] ShawnWiki at http://shawn.sourceforge.net/

[8] UZL Testbed Documentation at http://fronts.cti.gr/index.php/testbed

[9] Topology Recognition Application User Guide, online available at http://www.itm.uni-luebeck.de/projects/fronts/TopologyRecognitionApplication.zip?lang=de